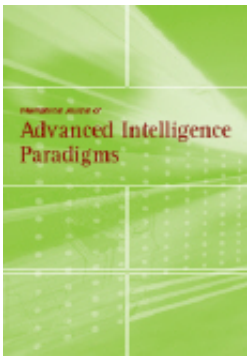


International Journal of Advanced Intelligence Paradigms

ISSN online: 1755-0394 - ISSN print: 1755-0386
https://www.inderscience.com/ijaip

Regression test case prioritisation using genetic algorithm

Anand Kumar Yadav, Anil Kumar Malviya

DOI: 10.1504/IJAIP.2024.10061366

Article History:

Received:	08 August 2018
Last revised:	24 August 2018
Accepted:	24 August 2018
Published online:	22 February 2024

Regression test case prioritisation using genetic algorithm

Anand Kumar Yadav*

Computer Science and Engineering Department,
Kamla Nehru Institute of Technology,
Sultanpur, Uttar Pradesh, India
Email: anandyadav2005@gmail.com
*Corresponding author

Anil Kumar Malviya

Computer Science and Engineering Department,
Kamla Nehru Institute of Technology,
Sultanpur, Uttar Pradesh, India
Email: anilkumarmalviya@gmail.com

Abstract: On customer's demand, new requirements are implemented in the software. The modified software may not work properly as earlier because of the new requirements added. So the modified software must be tested. Regression testing (RT) is defined as retesting of the modified software. It is performed using the already developed test suite and a newly developed test suite. The big software has a larger test suite size. For a single requirement change, to run the whole test cases is not beneficial for the development organisation. To make RT more effective, prioritisation of test suite is done. Here we present the genetic algorithm (GA) for the test case prioritisation (TCP). Different approaches have been discussed and implemented using the average percentage of fault detected (APFD) metric. The discussed approaches are applied over a single problem and the result is shown in the tabular form. APFD metric is applied to all the discussed approaches and suggested which one is better. This paper uses GA to arrange the test cases in a prioritised way on the basis of the fault detected.

Keywords: average percentage of fault detected; APFD; genetic algorithm; regression testing; test cases prioritisation; TCP.

Reference to this paper should be made as follows: Yadav, A.K. and Malviya, A.K. (2024) 'Regression test case prioritisation using genetic algorithm', *Int. J. Advanced Intelligence Paradigms*, Vol. 27, No. 1, pp.82–90.

Biographical notes: Anand Kumar Yadav received his BTech degree in Computer Science and Engineering from Uttar Pradesh Technical University (UPTU), Lucknow and MTech degree in Software Engineering from Gautam Buddha University (GBU), Greater Noida. He is currently pursuing his PhD in Computer Science and Engineering from Dr. A.P.J. Abdul Kalam Technical University, Lucknow, Uttar Pradesh, India. His areas of research interest cover software engineering, data mining, and computer networks.

Anil Kumar Malviya is currently working as a Professor and Head in Computer Science and Engineering Department at Kamla Nehru Institute of Technology (KNIT), Sultanpur, Uttar Pradesh, India. He received his BSc (Hons.) and MSc both in Computer Science from Banaras Hindu University (BHU), Varanasi, MTech degree in Computer Science and Engineering from KNIT, Sultanpur and his PhD degree in Computer Science from Dr. B.R. Ambedkar University, Agra. He is life time member of Computer Society of India (CSI) and Indian Society for Technical Education (ISTE). He has published about 55 papers in international/national journals, conferences and seminars. His research interests are software engineering, data mining, machine learning, and cryptography and network security. He guided research and produced PhDs in the field of computer science and engineering.

1 Introduction

There are some important phases in the software development process (SDP) such a requirement analysis phase, design phase, coding or implementation phase, and testing phase. After these four phases software become ready for use (Agarwal and Singh, 2008). The main purpose of the requirement phase is to collect requirement of the software from a customer and to document them in the proper way. A document is produced in the requirement phase which is known as software requirement specification (SRS) document. SRS document is transformed into a structure in the design phase. The produced structure is appropriate for implementation in some programming languages. In the coding phase, the code is written for the outcome of the design phase. During the coding test cases are written to test the code. All test cases are saved for future use when software is modified. Software testing team validates the software to meet the client's requirement before the final software is delivered to the customer. Software development process does not stop after the delivery of software. The last phase of SDP is known as the maintenance phase. This phase comes in picture when software is delivered at the client's site. Sometimes customers ask for the modification in the software in the form of addition or deletion of new functionality, requirements change and maybe some other. It becomes very difficult to modify existing software. It may be the release of a new version or a change in the existing version. Due to alteration in the existing software or reduction of functionality or addition of new functionality may lead to the introduction of the new faults in the software which causes software does not work properly. The maintenance phase ensures that software works properly.

In the maintenance phase, regression testing (RT) is performed. In the RT software is retested and make sure that there is no fault in the modified software. The RT ensures that no new error introduced in the modified software (Malishevsky et al., 2006).

To check the validity of the modified software tester writes some new test cases for the modified part and reuse the rest of test cases for unmodified part of the software which is saved during the SDP. In the RT, some test cases run instead of all test suites (Rothermel et al., 1997). The RT selects a minimum set of test cases which cover maximum code coverage. The selected test cases are minimised and prioritise to enhance the efficiency of the RT.

In this paper, we will use the genetic algorithm (GA). In most of the research work the GA used for the optimisation problem. It helps to reduce a large problem (more

computational time) on the basis of some constraints. With the help of GA, we can minimise the problem, prioritised the sub-problem of the problem and select the sub-problem which will reflect the whole problem. That is why this is an optimisation algorithm. There are three main constructs which play the vital role in GA are selection, crossover, and mutation (Balasubramanian and Manavalan, 2015; Ali and Mounir, 2016; Srinivas and Patnaik, 1994).

- *Problem statement* – the objective of the research paper is to make use of the GA for prioritisation of test cases on the basis of the fault detection.

This paper is organised in six sections: Section 2 represents work related to RT; Section 3 represents methodology; Section 4 shows the experimentation and analysis; Section 5 represents result analysis and Section 6 represents conclusion.

2 Related work

Some researcher paper has mentioned the problem of test case prioritisation (TCP) and proposed various models to solve it.

Srivastava (2008) suggested that the TCP according to the increasing value of average percentage of fault detected (APFD) metrics. In this paper, a technique was proposed. This technique calculates the average number of faults per minutes by a test case. The calculated value is used to arrange the test cases in decreasing order. For the calculation of the APFD metric fault must be known. This is the main disadvantages of the APFD metric.

Malangave and Kulkarni (2008) have discussed and implemented a test framework with an example of the software industry and suggested that this framework will reduce the effect of the RT in the software industry. Test framework is an extension of rational functional tester tool. Test framework works on the code coverage. The code coverage includes blocks, methods, and line of codes. The result depends on the different criteria, this is the disadvantage. When testing is performed on an instrumented version of the application, it reduces the performance of automated test execution. It is expensive because this technology is completely based upon code coverage.

Zheng et al. (2007) have discussed five TCP techniques. The result of an empirical study was presented. The result shows the comparative effectiveness of different techniques. The authors show that the overall performance of the additional greedy algorithm and 2-optimal algorithms are better than the greedy algorithm performs. The 2-optimal technique is better than greedy and additional greedy. The author depicts the advantages of both algorithms 2-optimal and the GA.

Rothermel et al. (2001) talked about TCP techniques for the RT. All techniques are experimentally examined to detect the fault. These techniques are based on code coverage. The experimental result suggested that the probability of fault exposing-based prioritisation performed is worse in the total branch coverage other than all coverage-based techniques. And the probability of fault exposing-based prioritisation in compare to coverage-based techniques may not be cost effective.

Elbaum et al. (2002) have done several empirical and case studies and find that a specific version of TCP improves the rate of fault detection. Kapfhammer and Soffa (2007) have presented coverage effectiveness metric to evaluate the TCP techniques on the basis of the coverage criterion and test quality is defined on the coverage criteria.

Rothermel et al. (1999) have experimentally verified that some techniques help to improve the rate of the fault detection.

Yadav and Malviya (2017) have presented a paper for test case generation using GA. In this paper, authors have taken triangle problem for an example to explain the test case generation process using GA. The GA is discussed step by step and control flow graph is drawn for the triangle problem. Control flow graph is drawn from the source code. The GA works better for larger test cases.

3 Methodology

In this paper, we will use the GA. It is an optimisation technique. Difficult problems are solved by GA up to an optimal or near optimal solution. Big problems can be solved easily with GA. The GA works on the principles of genetics and natural selection. In research, GA is mostly used in machine learning (Tutorialspoint, 2018). The population of the problem which is being solved known as the collection of chromosomes. A chromosome is a string of number, binary digits, characters, or symbols. And each representative in the chromosome's string is called a gene. The GA pseudocode is as follow (Goldberg, 1989):

1 The population of n test cases is generated randomly.

2 The fitness value is computed for each test case in the population.

3 Creation a new population:
   a Select two test cases with better fitness value.
   b To form a new test case with crossover probability.
   c Mutate new test case with mutation probability.
   d Put new generated test case into a new population.

4 For the further run of algorithms, a newly generated population is replaced.

5 Stop the execution and return the best solution if a defined condition is satisfied.

6 Go to step 2.

GA uses three operators-selection, crossover, and mutation on its population (Sharma et al., 2013).

- *Selection*: a selection process determines how two parents are nominated for mating based on their fitness value.
- *Crossove*r: it is applied to the selected chromosomes. Crossover means swapping of one gene or more consecutive sequence of genes in the string between two parents.
- *Mutatio*n: it is performed after the crossover. To keep genetic variety in the population, the mutation operator is used. One or more genes in the chromosome are altered by the mutation operator.

4 Experimental analysis

The APFD metric is given below.

$$\text{APFD} = 1 - \frac{\sum_{i=1}^{m} TFi}{n * m} + \frac{1}{2 * n}$$

Here n, m, and TFi are defined as follows: n represents the number of test cases and m represents the number of faults detected. The first test case in T (test suite) that exposes fault i is represented by TFi. From Table 1, no. of faults (m) = 8 and no. of test cases (n) = 8. So putting these values of m, n, and TFi in the above equation to calculate the APFD value.

Table 1 The number of faults (F_i) detected (d) by test cases (TC_i)

	F_1	F_2	F_3	F_4	F_5	F_6	F_7	F_8
TC_1	d							
TC_2			d		d			d
TC_3								d
TC_4		d				d	d	
TC_5			d				d	
TC_6	d		d					
TC_7		d		d				
TC_8		d			d			

1 Unordered TCP

Test case order – TC_1 TC_2 TC_3 TC_4 TC_5 TC_6 TC_7 TC_8.

$$\text{APFD} = 1 - \frac{1+4+2+7+2+4+4+2}{8*8} + \frac{1}{2*8} = 65.62\%$$

2 reverse order TCP

Test case order – TC_8 TC_7 TC_6 TC_5 TC_4 TC_3 TC_2 TC_1.

$$\text{APFD} = 1 - \frac{3+1+3+2+1+5+4+6}{8*8} + \frac{1}{2*8} = 67.18\%$$

3 Random order TCP

Test case random order – TC_3 TC_4 TC_2 TC_8 TC_6 TC_1 TC_5 TC_7.

$$\text{APFD} = 1 - \frac{5+2+3+8+3+2+2+1}{8*8} + \frac{1}{2*8} = 65.62\%$$

4 2-optimal algorithm prioritisation

Test case order – TC_2 TC_4 TC_1 TC_7 TC_3 TC_5 TC_6 TC_8.

$$\text{APFD} = 1 - \frac{3+2+1+4+1+2+2+1}{8*8} + \frac{1}{2*8} = 81.25\%$$

5 The genetic algorithm TCP

Test case order – TC_4 TC_2 TC_7 TC_1 TC_8 TC_6 TC_5 TC_3.

Last iteration step of the GA is:

- Population selection

TC_4	TC_2	TC_7	TC_8	TC_5	TC_6	TC_3	TC_1
TC_7	TC_8	TC_3	TC_2	TC_1	TC_4	TC_6	TC_5

- Crossover applied

TC_7	TC_8	TC_3	TC_2	TC_5	TC_6	TC_1	TC_4
TC_4	TC_2	TC_7	TC_8	TC_1	TC_6	TC_5	TC_3

- Mutation applied

TC_4	TC_2	TC_7	TC_1	TC_8	TC_6	TC_5	TC_3

$$\text{APFD} = 1 - \frac{4+1+2+3+2+1+1+2}{8*8} + \frac{1}{2*8} = 81.25\%$$

Table 2 APFD value of prioritised test suite

Techniques		*APFD (in percent)*
1	Non-prioritised	65.62
2	Reverse of non-prioritised	67.18
3	Random	65.62
4	2-optimal	81.25
5	Genetic algorithm	81.25

5 Result analysis

Table 2 represents the techniques and their corresponding APFD values. The comparison is drawn among GA, 2-optimal algorithm, random selection, non-prioritised, and reverse of the non-prioritised case. Which shows that both the GA and 2-optimal are better. Below graphs (Figure 1 to Figure 5) show the result of the above-mentioned algorithms and also the percentage of fault detected. Figure 1 represents the unordered test suite scheduled ($TC_1TC_2TC_3TC_4TC_5TC_6TC_7TC_8$) for execution. APFD metric for the unordered test suite is 65.62%. Figure 2 represents the reverse order test suite scheduled for execution. APFD metric for reverse order test suite is 67.18%. Figure 3 represents the random test suite scheduled for execution. ADFD metric for random test scheduled is 65.62%. Figure 4 represents the 2-optimal test suite scheduled for execution. APFD metric value for 2-optimal test suite schedule is 81.25%. Figure 5 represents the GA test suite schedule ($TC_4TC_2TC_7TC_1TC_8TC_6TC_5TC_3$) for execution. APFD metric value for the GA is 81.255%5. APFD metric values lie between 0 and 100. The value near 100 shows a better result. Figure 6 represents graph among techniques and their corresponding APFD metric values. Figures 1 to 6 show that 2-optimal and the GA have

better and same result. But for the larger value of the fault detected and test cases, the GA may give the better result.

Figure 1 Unordered test suite (see online version for colours)

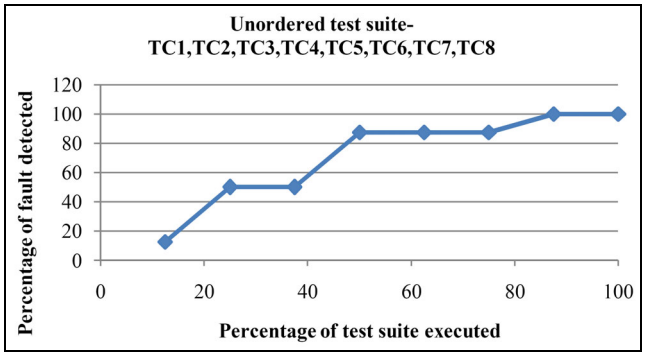


Figure 2 Reverse order test suite (see online version for colours)

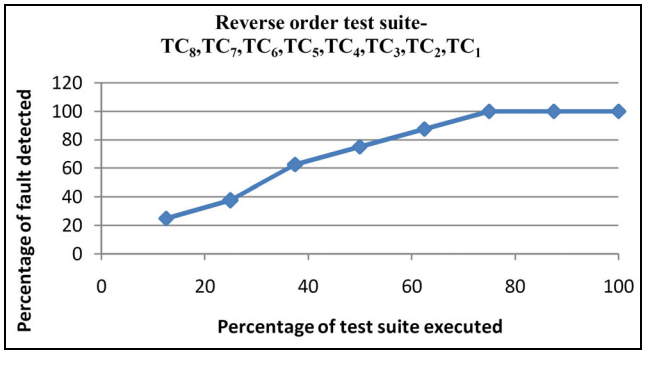


Figure 3 Random test suite (see online version for colours)

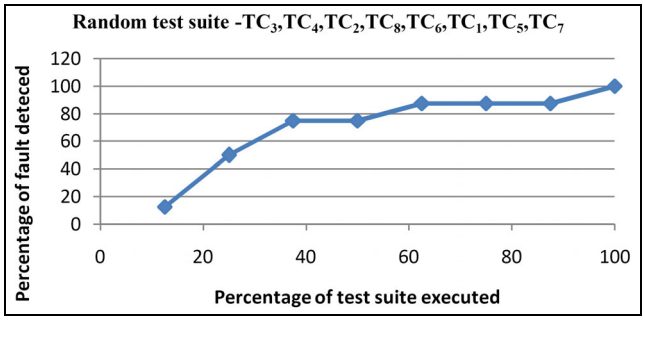


Figure 4 2-optimal test suite (see online version for colours)

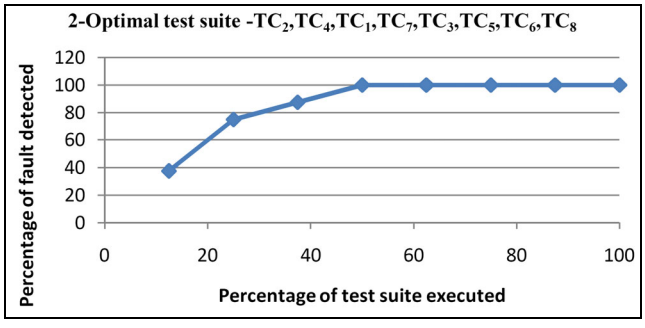


Figure 5 GA test suite (see online version for colours)

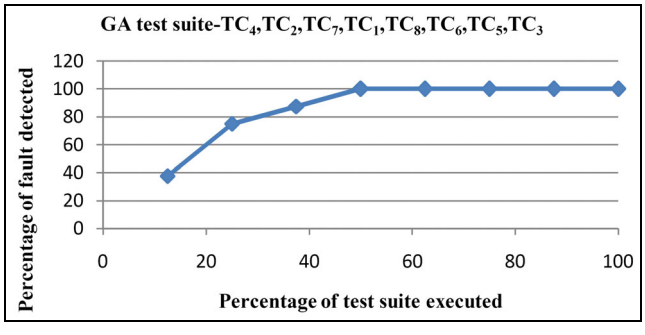
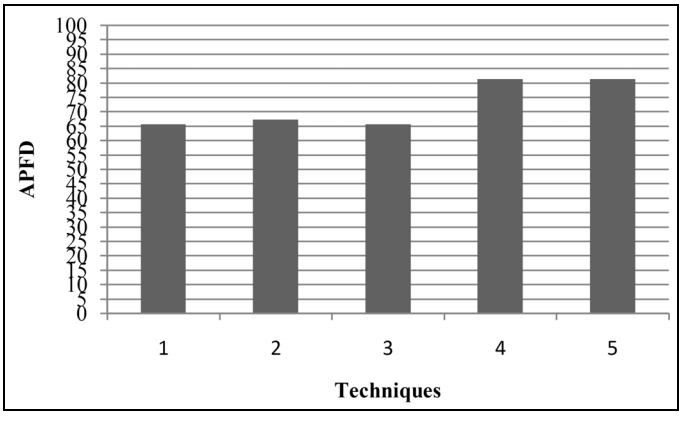


Figure 6 APSC value of prioritised test suite



6 Conclusions

In this paper, we have used five techniques for TCP: unordered, reverse ordered, random, 2-optimal, and the GA. In various papers, these techniques are executed individually but here we present a comparative analysis. In the first technique, we are not going to change the test cases sequences. The second technique uses just the reverse test case sequence of the first technique. In the third technique, random test cases are selected and put in a

sequence (first selected will execute first) for execution. The fourth technique selects the best two test cases are selected first and further two test cases are selected from the rest of the test cases and so on. And the fifth technique is the GA. This paper used GA for the TCP. It helps to improve the RT. The analysis is done among five techniques. The graph proves that two techniques out of five are better. But when the larger test suite is used then the GA shows the better result. The GA is more effective. In the future, maybe another metric used for prioritisation.

References

Agarwal, K.K. and Singh, Y. (2008) *Software Engineering*, 3rd ed., New Age International Publishers, New Delhi.

Ali, R. and Mounir, G. (2016) 'Adaptive probabilities of crossover and mutation in genetic algorithm for solving stochastic vehicle routing problem', *Int. J. Advanced Intelligence Paradigms*, Vol. 8, No. 3, pp.318–32.

Balasubramanian, V.K. and Manavalan, K. (2015) 'Differential evolution versus genetic algorithm in optimizing the quantization table for JPEG baseline algorithm', *Int. J. Advanced Intelligence Paradigms*, Vol. 7, No. 2, pp.111–135.

Elbaum, S., Malishevsky, A. and Rothermel, G. (2002)) 'Test case prioritization: a family of empirical studies', *IEEE Transactions on Software Engineering*, Vol. 28, No. 2, pp.159–182.

Goldberg, D.E (1989) *Genetic Algorithms: In Search, Optimization and Machine Learning*, Addison-Wesley Longman Publishing Co., Inc. Boston, MA, USA, ISBN: 0201157675.

Kapfhammer, G.M. and Soffa, M.L. (2007) 'Using coverage effectiveness to evaluate test suit prioritizations', *Proceedings of the ACM International Workshop on Empirical Assessment of Software Engineering Languages and Technologies*, Atlanta, Georgia, pp.19–20.

Malangave, P. and Kulkarni, D.B. (2008) *Efficient Test Case Prioritization in Regression Testing* [online] http://www.academia.edu/705374/Efficient_Test_Case_Prioritization_in_Regression_Testing (accessed July 2018).

Malishevsky, A.G., Ruthruff, J.R., Rothermel, G. and Elbaum, S. (2006) *Cost-Cognizant Test Case Prioritization*, Technical Report TR-UNL-CSE-2006-0004, Department of Computer Science and Engineering, University of Nebraska-Lincoln, Nebraska, USA.

Rothermel, G., Untch, R.H. and Harrold, M.J. (2001) 'Prioritizing test cases for regression testing', *IEEE Transactions on Software Engineering*, Vol. 27, No. 10, pp.929–948.

Rothermel, G., Untch, R.H. and Rothermel, M.J. (1997) 'A safe efficient regression test selection technique', *ACM Trans. Software Eng. Methodology (TOSEM)*, Vol. 6, No. 2, pp.173–210.

Rothermel, G., Untch, R.H., Chu, C. and Harrold, M.J. (1999) 'Test case prioritization: an empirical study', *Proceedings of the International Conference on Software Maintenance*, Oxford, UK, pp.179–188.

Sharma, C., Sabharwal, S. and Sibal, R. (2013) 'A survey on software testing techniques using genetic algorithm', *IJCSI International Journal of Computer Science Issues*, Vol. 10, No. 1, pp.381–393.

Srinivas, M. and Patnaik, L.M. (1994) 'Adaptive probabilities of crossover and mutation in genetic algorithms', *IEEE Transactions on Systems, Man and Cybernetics*, Vol. 24, No. 4, pp.656–667.

Srivastava, P.R. (2008) 'Test case prioritization', *Journal of Theoretical and Applied Information Technology*, JATIT, pp.178–181.

Tutorialspoint (2018) [online] https://www.tutorialspoint.com/genetic_algorithms/genetic_algorithms_quick_guide.htm (accessed June 2018).

Yadav, A.K. and Malviya, A.K. (2017) 'Systematic test case generation using genetic algorithm', *International Conference on Innovative Entrepreneurship and Startup* (ICIES), pp.195–197, ISBN: 978-93-86256-55-3.

Zheng, L., Harman, M. and Hierons, R.M. (2007) 'Search algorithms for regression test case prioritization', *IEEE Transactions on Software Engineering*, Vol. 33, No. 4, pp.225–237.